cnrdmrci / **PL-SQL-CheatSheet** Public

PL/SQL-CheatSheet

14 stars 3 forks Branches Tags Activity

Star Notifications

Code Issues Pull requests Actions Projects Security Insights

master 1 Branch 0 Tags Go to file Go to file Code ...

cnrdmrci edit 4 years ago

README.md edit 4 years ago

Oracle PL/SQL(Procedural Language / Standard Query Language) CheatSheet

Contents

Blocks

```
SET SERVEROUTPUT ON;
DECLARE
 --Declaration statements

BEGIN
 --Executable statements

Exceptions
```

```
    --Exception handling statements

  END;
```

Variables

Data Types

- Scalar

> Number, Date, Boolean, Character

- Large Object

> Large Text, Picture - BFILE, BLOB, CLOB, NCLOB

- Composite

> Collections, Records

- Reference

```
--NUMBER(precision,scale)
v_number NUMBER(5,2) := 5.01;
v_character VARCHAR2(20) := 'test';
newyear DATE:='01-JAN-2020';
current_date DATE:=SYSDATE;
```

Constant

```
DECLARE
        v_pi CONSTANT NUMBER(7,6) := 3.141592;
BEGIN
        DBMS_OUTPUT.PUT_LINE(v_pi);
END;
```

Select Into

```
DECLARE
        v_last_name VARCHAR2(20);
BEGIN
        SELECT last_name INTO v_last_name FROM persons WHERE person_id = 1;
        DBMS_OUTPUT.PUT_LINE('Last name: ' || v_last_name);
END;
```

%Type

```
DECLARE
        v_last_name persons.last_name%TYPE;
BEGIN
        SELECT last_name INTO v_last_name FROM persons WHERE person_id = 1;
        DBMS_OUTPUT.PUT_LINE('Last Name: ' || v_last_name);
END;
```

Conditions

```
DECLARE
        v_num NUMBER := &enter_a_number;
```

```
BEGIN
        IF mod(v_num,2) = 0 THEN
            dbms_output.put_line(v_num || ' is even');
        ELSIF mod(v_num,2) = 1 THEN
            dbms_output.put_line(v_num || ' is odd');
        ELSE
            dbms_output.put_line('None');
        END IF;
END;
```

Case

```
set serveroutput on;
DECLARE
    a NUMBER :=65;
    b NUMBER :=2;
    arth_operation VARCHAR2(20) :='MULTIPLY';
BEGIN
    dbms_output.put_line('Program started.' );
    CASE (arth_operation)
        WHEN 'ADD' THEN
            dbms_output.put_line('Addition of the numbers are: '|| a+b );
        WHEN 'SUBTRACT' THEN
            dbms_output.put_line('Subtraction of the numbers are: '||a-b );
        WHEN 'MULTIPLY' THEN
            dbms_output.put_line('Multiplication of the numbers are: '|| a*b);
        WHEN 'DIVIDE' THEN
            dbms_output.put_line('Division of the numbers are:'|| a/b);
        ELSE
            dbms_output.put_line('No operation action defined. Invalid operation');
    END CASE;
    dbms_output.put_line('Program completed.' );
END;

--Searched case
DECLARE
    a NUMBER :=70;
    b NUMBER :=2;
    arth_operation VARCHAR2(20) :='DIVIDE';
BEGIN
    dbms_output.put_line('Program started.' );
    CASE
        WHEN arth_operation = 'ADD' THEN
            dbms_output.put_line('Addition of the numbers are: '||a+b );
        WHEN arth_operation = 'SUBTRACT' THEN
            dbms_output.put_line('Subtraction of the numbers are: '|| a-b);
        WHEN arth_operation = 'MULTIPLY' THEN
            dbms_output.put_line('Multiplication of the numbers are: '|| a*b );
        WHEN arth_operation = 'DIVIDE' THEN
            dbms_output.put_line('Division of the numbers are: '|| a/b );
        ELSE
            dbms_output.put_line('No operation action defined. Invalid operation');
    END CASE;
    dbms_output.put_line('Program completed.' );
END;
```

Loops

```
--Simple Loop
DECLARE
        v_num number(5) := 0;
BEGIN
        loop
            v_num := v_num + 1;
            dbms_output.put_line('Number: ' || v_num);

            exit when v_num = 3;
            /*
            if v_num = 3 then
                exit;
            end if;
            */
        end loop;
END;

--While Loop
DECLARE
        v_num number := 0;
BEGIN
        while v_num <= 100 loop

            exit when v_num > 40;

            if v_num = 20 then
                v_num := v_num + 1;
                continue;
            end if;

            if mod(v_num,10) = 0 then
                dbms_output.put_line(v_num || ' can be divided by 10.');
            end if;

            v_num := v_num + 1;

        end loop;
END;

--For Loop
DECLARE
        v_num number := 0;
BEGIN
        for x in 10 .. 13 loop
            dbms_output.put_line(x);
        end loop;

        for x in reverse 13 .. 15 loop
            if mod(x,2) = 0 then
                dbms_output.put_line('even: ' || x);
            else
                dbms_output.put_line('odd: ' || x);
            end if;
        end loop;
END;
```

Triggers

```
-- DML Triggers
CREATE OR REPLACE TRIGGER tr_persons
BEFORE INSERT OR DELETE OR UPDATE ON persons
FOR EACH ROW
ENABLE
DECLARE
        v_user varchar2(20);
BEGIN
        SELECT user INTO v_user FROM dual;
        IF INSERTING THEN
                DBMS_OUTPUT.PUT_LINE('One line inserted by ' || v_user);
        ELSIF DELETING THEN
                DBMS_OUTPUT.PUT_LINE('One line Deleted by ' || v_user);
        ELSIF UPDATING THEN
```

```
                DBMS_OUTPUT.PUT_LINE('One line Updated by ' || v_user);
        END IF;
END;

--

CREATE OR REPLACE TRIGGER persons_audit
BEFORE INSERT OR DELETE OR UPDATE ON persons
FOR EACH ROW
ENABLE
DECLARE
  v_user varchar2 (30);
  v_date  varchar2(30);
BEGIN
  SELECT user, TO_CHAR(sysdate, 'DD/MON/YYYY HH24:MI:SS') INTO v_user, v_date  FROM dual;
  IF INSERTING THEN
    INSERT INTO sh_audit (new_name,old_name, user_name, entry_date, operation)
    VALUES(:NEW.LAST_NAME, Null , v_user, v_date, 'Insert');
  ELSIF DELETING THEN
    INSERT INTO sh_audit (new_name,old_name, user_name, entry_date, operation)
    VALUES(NULL,:OLD.LAST_NAME, v_user, v_date, 'Delete');
  ELSIF UPDATING THEN
    INSERT INTO sh_audit (new_name,old_name, user_name, entry_date, operation)
    VALUES(:NEW.LAST_NAME, :OLD.LAST_NAME, v_user, v_date,'Update');
  END IF;
END;

-- DDL Triggers
CREATE OR REPLACE TRIGGER db_audit_tr
AFTER DDL ON DATABASE
BEGIN
    INSERT INTO schema_audit VALUES (
                sysdate,
                sys_context('USERENV','CURRENT_USER'),
                ora_dict_obj_type,
                ora_dict_obj_name,
                ora_sysevent);
END;

-- Instead of Triggers
CREATE VIEW vw_twotable AS
SELECT full_name, subject_name FROM persons, subjects;

CREATE OR REPLACE TRIGGER tr_Insert
INSTEAD OF INSERT ON vw_twotable
FOR EACH ROW
BEGIN
  INSERT INTO persons (full_name) VALUES (:new.full_name);
  INSERT INTO subjects (subject_name) VALUES (:new.subject_name);
END;

insert into vw_twotable values ('Caner','subject');
```

Cursors

```
--%FOUND
--%NOTFOUND
--%ISOPEN
--%ROWCOUNT

declare
    v_first_name varchar2(20);
    v_last_name varchar2(20);
    Cursor test_cursor is select first_name,last_name from persons;
begin
    open test_cursor;
    loop
        fetch test_cursor into v_first_name,v_last_name;
        exit when test_cursor%NOTFOUND;
        dbms_output.put_line('Name: ' || v_first_name || ', Lastname: ' || v_last_name);
    end loop;
    close test_cursor;
end;

----

declare
    v_first_name varchar2(20);
    v_last_name varchar2(20);
    Cursor test_cursor (first_name_parameter varchar2) is
        select first_name,last_name from persons where first_name = first_name_parameter;
begin
    open test_cursor('caner');
    loop
        fetch test_cursor into v_first_name,v_last_name;
        exit when test_cursor%NOTFOUND;
        dbms_output.put_line('Name: ' || v_first_name || ', Lastname: ' || v_last_name);
    end loop;
    close test_cursor;
end;

----

declare
    v_first_name varchar2(20);
    v_last_name varchar2(20);
    Cursor test_cursor (first_name_parameter varchar2 := 'caner') is
        select first_name,last_name from persons where first_name = first_name_parameter;
begin
    open test_cursor;
    loop
        fetch test_cursor into v_first_name,v_last_name;
        exit when test_cursor%NOTFOUND;
        dbms_output.put_line('Name: ' || v_first_name || ', Lastname: ' || v_last_name);
    end loop;
    close test_cursor;
end;

--for
declare
    Cursor test_cursor is select first_name,last_name from persons;
begin
    for obj in test_cursor
    loop
        dbms_output.put_line('Name: ' || obj.first_name || ', Lastname: ' || obj.last_name);
    end loop;
end;

--for parameter
declare
    Cursor test_cursor (first_name_parameter varchar2 := 'can') is
        select first_name,last_name from persons where first_name = first_name_parameter;
begin
    for obj in test_cursor('caner')
    loop
        dbms_output.put_line('Name: ' || obj.first_name || ', Lastname: ' || obj.last_name);
    end loop;
end;
```

Records

```
--table based
declare
    v_person persons%ROWTYPE;
begin
    select * into v_person from persons where PERSON_ID = 2;
    dbms_output.put_line('Name: ' || v_person.first_name || ', Lastname: ' || v_person.last_name);
end;

--

declare
    v_person persons%ROWTYPE;
begin
    select first_name,last_name into v_person.first_name,v_person.last_name
        from persons where PERSON_ID = 2;
    dbms_output.put_line('Name: ' || v_person.first_name || ', Lastname: ' || v_person.last_name);
end;

--cursor based record
declare
    Cursor test_cursor is select first_name,last_name from persons where person_id = 2;
    v_person test_cursor%rowtype;
begin
    open test_cursor;
    fetch test_cursor into v_person;
    dbms_output.put_line('Name: ' || v_person.first_name || ', Lastname: ' || v_person.last_name);
    close test_cursor;
end;

--

declare
    Cursor test_cursor is select first_name,last_name from persons;
    v_person test_cursor%rowtype;
begin
    open test_cursor;
    loop
        fetch test_cursor into v_person;
        exit when test_cursor%NOTFOUND;
        dbms_output.put_line('Name: ' || v_person.first_name || ', Lastname: ' || v_person.last_name);
    end loop;
    close test_cursor;
end;

--user based

declare
    type rv_person is record(
        f_name varchar2(20),
        l_name persons.last_name%type
    );
    v_person rv_person;
begin
    select first_name,last_name into v_person.f_name,v_person.l_name from persons where person_id = 2;
    dbms_output.put_line('Name: ' || v_person.f_name || ', Lastname: ' || v_person.l_name);
end;
```

Functions

```
CREATE OR REPLACE FUNCTION circle_area (radius NUMBER)
RETURN NUMBER IS
--Declare a constant and a variable
pi      CONSTANT NUMBER(7,3) := 3.141;
area    NUMBER(7,3);
BEGIN
```

```
 --Area of Circle pi*r*r;
 area := pi * (radius * radius);
 RETURN area;
END;

BEGIN
    dbms_output.put_line('Alan: ' || circle_area(10));
END;
```

Stored Procedure

```
create or replace procedure pr_test is
  v_name varchar(20) := 'Caner';
  v_city varchar(20) := 'Istanbul';
begin
  dbms_output.put_line(v_name || ',' || v_city);
end pr_test;
--
execute pr_test;
--
begin
  pr_test;
end;

----

create or replace procedure pr_test_param(v_name varchar2 default 'caz')
is
  v_city varchar(20) := 'Istanbul';
begin
  dbms_output.put_line(v_name || ',' || v_city);
end pr_test_param;
--
execute pr_test_param(v_name => 'cam');

----

create or replace procedure pr_test_param(v_name varchar2)
is
  v_city varchar(20) := 'Istanbul';
begin
  dbms_output.put_line(v_name || ',' || v_city);
end pr_test_param;
--
execute pr_test_param('Caner');
--
begin
  pr_test_param('Caner');
end;
```

Package

```
CREATE OR REPLACE PACKAGE pkg_person IS
 FUNCTION get_name (v_name VARCHAR2) RETURN VARCHAR2;
 PROCEDURE proc_update_lastname(p_id NUMBER, l_name VARCHAR2);
END pkg_person;

--Package Body
CREATE OR REPLACE PACKAGE BODY pkg_person IS
 --Function Implimentation
 FUNCTION get_name (v_name VARCHAR2) RETURN VARCHAR2 IS
  BEGIN
   RETURN v_name;
  END get_name;
  --Procedure Implimentation
   PROCEDURE proc_update_lastname(p_id NUMBER, l_name VARCHAR2) IS
     BEGIN
      UPDATE persons SET last_name = l_name where person_id = p_id;
     END;

END pkg_person;

--
begin
    dbms_output.put_line(pkg_person.get_name('Caner'));
end;
execute pkg_person.proc_update_lastname(2,'new lastname');
```

Exceptions

```
accept p_divisor number prompt 'Enter divisor';
declare
    v_divided number := 24;
    v_divisor number := &p_divisor;
    v_result number;
    ex_four exception;
    pragma exception_init(ex_four,-20001); --20000 , 20999
begin
    if v_divisor = 4 then
        raise ex_four;
    end if;

    if v_divisor = 5 then
        raise_application_error(-20001,'div five');
    end if;

    if v_divisor = 6 then
        raise_application_error(-20002,'div six');
    end if;

    v_result := v_divided/v_divisor;

    exception
        when ex_four then  --user defined
            dbms_output.put_line('Div four');
            dbms_output.put_line(SQLERRM);
        when ZERO_DIVIDE then --system defined
            dbms_output.put_line('Div zero');
        when OTHERS then
            dbms_output.put_line('Other exception');
            dbms_output.put_line(SQLERRM);
end;
```

Collections

```
--Nested table
DECLARE
  TYPE my_nested_table IS TABLE OF number;
  var_nt  my_nested_table :=  my_nested_table (5,12,17,66,44,88,25,45,65);
BEGIN
  FOR i IN 1..var_nt.COUNT
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Value stored at index '||i||' is '||var_nt(i));
  END LOOP;
END;

--VARRAY
```

```
DECLARE
    TYPE inBlock_vry IS VARRAY (5) OF NUMBER;
    vry_obj inBlock_vry  :=  inBlock_vry(); --inBlock_vry(null,null,null,null,null);
BEGIN
    --vry_obj.EXTEND(5);
    FOR i IN 1 .. vry_obj.LIMIT
    LOOP
        vry_obj.EXTEND;
        vry_obj (i):= 10*i;
        DBMS_OUTPUT.PUT_LINE (vry_obj (i));
    END LOOP;
END;

--Associative Array(dictionary)
DECLARE
    TYPE books IS TABLE OF NUMBER INDEX BY VARCHAR2 (20);
    Isbn Books;
BEGIN
        -- How to insert data into the associative array
        isbn('Oracle Database') := 1122;
        isbn('MySQL') := 6543;
        DBMS_OUTPUT.PUT_LINE('Value Before Updation '||isbn('MySQL'));

        -- How to update data of associative array.
        isbn('MySQL') := 2222;

        -- how to retrieve data using key from associative array.
        DBMS_OUTPUT.PUT_LINE('Value After Updation '||isbn('MySQL'));
END;

--

DECLARE
    TYPE books IS TABLE OF NUMBER INDEX BY VARCHAR2 (20);
    Isbn Books;
    flag varchar2(20);
BEGIN
    isbn('Oracle Database') := 1122;
    isbn('MySQL') := 6543;
    isbn('MySQL') := 2222;
    flag := isbn.FIRST;
    while flag is not null
    loop
        DBMS_OUTPUT.PUT_LINE('Key -> '||flag||'Value -> '||isbn(flag));
        flag := isbn.NEXT(flag);
    end loop;
END;

-----Collection Methods
--Count
DECLARE
    TYPE my_nested_table IS TABLE OF number;
    var_nt my_nested_table := my_nested_table (5,12,17,66,44,88,25,45,65);
BEGIN
    DBMS_OUTPUT.PUT_LINE ('The Size of the Nested Table is ' ||var_nt.count);
END;


--exists
DECLARE
    TYPE my_nested_table IS TABLE OF VARCHAR2 (20);
        col_var_1   my_nested_table := my_nested_table('Super Man','Iron Man','Bat Man');
BEGIN
    IF col_var_1.EXISTS (4) THEN
        DBMS_OUTPUT.PUT_LINE ('Hey we found '||col_var_1 (1));
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Sorry, no data at this INDEX');
        col_var_1.EXTEND;
        col_var_1(4) := 'Spiderman';
    END IF;
    IF col_var_1.EXISTS (4) THEN
        DBMS_OUTPUT.PUT_LINE ('New data at index 4 '||col_var_1 (4));
    end if;
END;

--first and last
SET SERVEROUTPUT ON;
DECLARE
    TYPE nt_tab IS TABLE OF NUMBER;
    col_var nt_tab := nt_tab(10, 20, 30, 40, 50);
BEGIN
    col_var.DELETE(1);
    col_var.TRIM;
    DBMS_OUTPUT.PUT_LINE ('First Index of the Nested table is ' || col_var.FIRST);
    DBMS_OUTPUT.PUT_LINE ('Last Index of the Nested table is ' || col_var.LAST);

    DBMS_OUTPUT.PUT_LINE ('Value stored at First Index is ' || col_var(col_var.FIRST));
    DBMS_OUTPUT.PUT_LINE ('Value stored at First Index is ' || col_var(col_var.LAST));
END;

--limit
DECLARE
    TYPE inBlock_vry IS VARRAY (5) OF NUMBER;
    vry_obj inBlock_vry := inBlock_vry();
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Total Indexes '||vry_obj.LIMIT);
END;

--
DECLARE
    --Create VARRAY of 5 element
    TYPE inblock_vry IS
        VARRAY ( 5 ) OF NUMBER;
    vry_obj   inblock_vry := inblock_vry ();
BEGIN
    vry_obj.extend;
    vry_obj(1) := 10 * 2;
    dbms_output.put_line('Total Number of Index ' || vry_obj.limit);
    dbms_output.put_line('Total Number of Index which are occupied ' || vry_obj.count);
END;

-- Prior and Next
DECLARE
    TYPE my_nested_table IS
        TABLE OF NUMBER;
    var_nt   my_nested_table := my_nested_table(5,12,17,66,44,88,25,45,65);
BEGIN
        var_nt.DELETE(2);
        dbms_output.put_line('Index prior to index 3 is '||var_nt.PRIOR(3));
        dbms_output.put_line('Value before 3rd Index is '||var_nt(var_nt.PRIOR(3)));
END;
--
DECLARE
    TYPE my_nested_table IS
        TABLE OF NUMBER;
    var_nt   my_nested_table := my_nested_table(5,12,17,66,44,88,25,45,65);
BEGIN
        dbms_output.put_line('Next Higher Index to index 3 is '||var_nt.NEXT(3));
        dbms_output.put_line('Value after 3rd Index is '||var_nt(var_nt.NEXT(3)));
END;

--Delete
DECLARE
    TYPE my_nested_table IS
        TABLE OF NUMBER;
    var_nt my_nested_table := my_nested_table(2,4,6,8,10,12,14,16,18,20);
```

```
BEGIN

    --Delete Range
    var_nt.DELETE(2,6);
    FOR i IN 1..var_nt.LAST LOOP
        IF var_nt.EXISTS(i) THEN
            DBMS_OUTPUT.PUT_LINE('Value at Index ['||i||'] is '|| var_nt(i));
        END IF;
    END LOOP;
END;

--extend
DECLARE
    TYPE my_nestedTable IS TABLE OF number;
    nt_obj  my_nestedTable := my_nestedTable();
BEGIN
    nt_obj.EXTEND;
    nt_obj(1) := 28;
    nt_obj.EXTEND(3);
    nt_obj(2) := 10;
    nt_obj(3) := 20;
    nt_obj(4) := 30;
    DBMS_OUTPUT.PUT_LINE ('Data at index 1 is '||nt_obj(1));
    DBMS_OUTPUT.PUT_LINE ('Data at index 2 is '||nt_obj(2));
    DBMS_OUTPUT.PUT_LINE ('Data at index 3 is '||nt_obj(3));
    DBMS_OUTPUT.PUT_LINE ('Data at index 4 is '||nt_obj(4));
    nt_obj.EXTEND(2,4);
    DBMS_OUTPUT.PUT_LINE ('Data at index 5 is '||nt_obj(5));
    DBMS_OUTPUT.PUT_LINE ('Data at index 6 is '||nt_obj(6));
END;

--TRIM
DECLARE
 TYPE inBlock_vry IS VARRAY (5) OF NUMBER;
 vry_obj inBlock_vry := inBlock_vry(1, 2, 3, 4, 5);
BEGIN
    --TRIM without parameter
    vry_obj.TRIM;
    DBMS_OUTPUT.PUT_LINE ('After TRIM procedure');
    FOR i IN 1..vry_obj.COUNT
    LOOP
        DBMS_OUTPUT.PUT_LINE (vry_obj(i));
    END LOOP;
    --TRIM with Parameter
    vry_obj.TRIM (2);
    DBMS_OUTPUT.PUT_LINE ('After TRIM procedure');
    FOR i IN 1..vry_obj.COUNT
    LOOP
        DBMS_OUTPUT.PUT_LINE (vry_obj(i));
    END LOOP;
END;
--
DECLARE
    TYPE my_nestedTable IS TABLE OF number;
    nt_obj  my_nestedTable := my_nestedTable(1,2,3,4,5);
BEGIN
    nt_obj.TRIM (3);
    DBMS_OUTPUT.PUT_LINE ('After TRIM procedure');
    FOR i IN 1..nt_obj.COUNT
    LOOP
        DBMS_OUTPUT.PUT_LINE (nt_obj(i));
    END LOOP;
END;
```

Object Oriented

```
CREATE OR REPLACE TYPE Worker AS OBJECT (
```

README

```
        v_last_name varchar(10),
        v_email varchar(20),
        member procedure display,
        member function getName return varchar2,
        static procedure displaySquare(v_num number)
    );

    CREATE OR REPLACE TYPE BODY Worker AS
        MEMBER PROCEDURE display IS
        BEGIN
            DBMS_OUTPUT.put_line('id: '||SELF.v_id);
            DBMS_OUTPUT.put_line('name: '||SELF.v_name);
            DBMS_OUTPUT.put_line('lastName : '||SELF.v_last_name);
            DBMS_OUTPUT.put_line('mail: '||SELF.v_email);
        END;
        MEMBER FUNCTION getName RETURN VARCHAR2 IS
        BEGIN
            RETURN SELF.v_name || ' ' || SELF.v_last_name;
        END;
        STATIC PROCEDURE displaySquare(v_num number) IS
        BEGIN
            DBMS_OUTPUT.put_line('Square : '||v_num);
        END;
    END;

    DECLARE
        v_person Worker := new Worker(1, 'Caner', 'lastName', 'mail@.com'); --constructor
    BEGIN
        DBMS_OUTPUT.put_line('Name: '||v_person.getName());
```

Releases

No releases published

Packages

No packages published